Parse Tree

PROGRAM

```
<p4program> := <declarationList>

<declarationList> := { <declaration> | ; /* empty declaration */ }*

<declaration> := <variableDeclaration> | <externDeclaration> | <actionDeclaration> |
  <typeDeclaration> | <parserDeclaration> | <controlDeclaration> | <instantiation> |
  <errorDeclaration> | <matchKindDeclaration> | <functionDeclaration>

<nonTypeName> := IDENTIFIER | apply | key | actions | state | entries | type

<name> := <nonTypeName> | TYPE_IDENTIFIER

<nonTableKwName> := IDENTIFIER | TYPE_IDENTIFIER | apply | state | type

<parameterList> := { <parameter> { , <parameter> }* }

<parameter> := { <direction> } <typeRef> <name> { = <expression> }

<direction> := in | out | inout

<packageTypeDeclaration> := package <name> ~~<optTypeParameters>~~ ( <parameterList> )

<instantiation> := <typeRef> ( <argumentList> ) <name> ;

<optConstructorParameters> := { ( <parameterList> ) }
```

PARSER

```
<parserDeclaration> := <parserTypeDeclaration> <optConstructorParameters>
  { <parserLocalElements> <parserStates> }

<parserLocalElements> := { <parserLocalElement> }*

<parserLocalElement> := <variableDeclaration> |  <instantiation> /* | <valueSetDeclaration> */

<parserTypeDeclaration> := parser <name> <optTypeParameters> ( <parameterList> )

<parserStates> := <parserState> { <parserState> }*

<parserState> := state <name> { <parserStatements> <transitionStatement> }

<parserStatements> := { <parserStatement> }*

<parserStatement> := <assignmentOrMethodCallStatement> | <directApplication> | <parserBlockStatement> |
  <variableDeclaration> | <emptyStatement>

<parserBlockStatement> := { <parserStatements> }

<transitionStatement> := { transition <stateExpression> }

<stateExpression> := <name> ; | <selectExpression>

<selectExpression> := select ( <expressionList> ) { <selectCaseList> }

<selectCaseList> := { <selectCase> }*

<selectCase> := <keysetExpression> : <name> ;

<keysetExpression> := <tupleKeysetExpression> | <simpleKeysetExpression>

<tupleKeysetExpression> := ( <simpleExpressionList> )

<simpleExpressionList> := <simpleKeysetExpression> { , <simpleKeysetExpression }*

<simpleKeysetExpression> := <expression> /* { ( mask | range ) <expression> } */ | default | _

/* <valueSetDeclaration> := valueset '<' ( <baseType> | <tupleType> | <typeName> ) '>'
  '(' <expression> ')' <name> ';' */
```

CONTROL

```
<controlDeclaration> := <controlTypeDeclaration> <optConstructorParameters>
  { <controlLocalDeclarations> apply <blockStatement> }

<controlTypeDeclaration> := control <name> <optTypeParameters> ( <parameterList> )

<controlLocalDeclaration> := <variableDeclaration> | <actionDeclaration> | <tableDeclaration> |
  <instantiation>

<controlLocalDeclarations> := { <controlLocalDeclaration }*
```

EXTERN

```
<externDeclaration> := extern
  ( <nonTypeName> <optTypeParameters> { <methodPrototypes> } ) |
  ( <functionPrototype> ; )

<methodPrototypes> := { <methodPrototype> }*

<functionPrototype> := <typeOrVoid> <name> <optTypeParameters> ( <parameterList> )

<methodPrototype> :=
  ( <functionPrototype> ; ) |
  ( TYPE ( <parameterList> ) ; )  /* constructor */
```

TYPES

```
<typeRef> := <baseType> | <namedType> | <tupleType>

<namedType> := <typeName> | ~~<specializedType>~~ | <headerStackType>

<typeName> := TYPE_IDENTIFIER

<tupleType> := tuple < <typeArgumentList> >

<headerStackType> := <typeName> [ <expression> ]

~~<specializedType> := <typeName> < <typeArgumentList> >~~

<baseType> :=
  bool | error | string | void |
  int { < <integerTypeSize> > } |
  bit { < <integerTypeSize> > } |
  varbit < <integerTypeSize> >

<integerTypeSize> := INTEGER /* | '(' <expression> ')' */

<typeOrVoid> := <typeRef> | void | IDENTIFIER  /* type variable */

~~<optTypeParameters> := { < <typeParameterList> > }~~

~~<typeParameterList> := <name> { , <name> }*~~

<realTypeArg> := _ | <typeRef>

<typeArg> := _ | <typeRef> | <nonTypeName>

<realTypeArgumentList> := <realTypeArg> { , <realTypeArg> }*

<typeArgumentList> := { <typeArg> { , <typeArg> }* }

<typeDeclaration> := <derivedTypeDeclaration> | <typedefDeclaration> | ( <parserTypeDeclaration> ; )
  | ( <controlTypeDeclaration> ; ) | ( <packageTypeDeclaration> ; )

<derivedTypeDeclaration> := <headerTypeDeclaration> | <headerUnionDeclaration> |
  <structTypeDeclaration> | <enumDeclaration>

<headerTypeDeclaration> := header <name> { <structFieldList> }

<headerUnionDeclaration> := header_union <name> { <structFieldList> }

<structTypeDeclaration> := struct <name> { <structFieldList> }

<structFieldList> := { <structField> { , <structField> }* }

<structField> := <typeRef> <name> ;

<enumDeclaration> := enum { bit < INTEGER > } <name> { <specifiedIdentifierList> }

<errorDeclaration> := error { <identifierList> }

<matchKindDeclaration> := match_kind { <identifierList> }

<identifierList> := <name> { , <name> }*
```

```
<specifiedIdentifierList> := <specifiedIdentifier> { , <specifiedIdentifier> }*

<specifiedIdentifier> := <name> { = <expression> }

<typedefDeclaration> := ( typedef | ~~type~~ ) ( <typeRef> | <derivedTypeDeclaration> ) <name> ';'
```

STATEMENTS

```
<assignmentOrMethodCallStatement> := <lvalue>
  ( ~~{ < <typeArgumentList> > }~~ ( <argumentList> ) ; ) |
  ( = <expression> ; )

<emptyStatement> := ;

<returnStatement> := return { expression } ;

<exitStatement> := exit ;

<conditionalStatement> := if ( <expression> ) <statement> { else <statement> }

/* To support direct invocation of a control or parser without instantiation. */
<directApplication> := <typeName> . apply ( <argumentList> ) ;

<statement> := <assignmentOrMethodCallStatement> | <directApplication> | <conditionalStatement> |
  <emptyStatement> | <blockStatement> | <exitStatement> | <returnStatement> | <switchStatement>

<blockStatement> := { <statementOrDeclList> }

<statementOrDeclList> := { <statementOrDeclaration> }*

<switchStatement> := switch ( <expression> ) { <switchCases> }

<switchCases> := { <switchCase> }*

<switchCase> := <switchLabel> : { <blockStatement> }

<switchLabel> := <name> | default

<statementOrDeclaration> := <variableDeclaration> | <statement> | <instantiation>
```

TABLES

```
<tableDeclaration> := table <name> { <tablePropertyList> }

<tablePropertyList> := <tableProperty> { <tableProperty> }*

<tableProperty> :=
  ( key = { <keyElementList> } ) |
  ( actions = { <actionList> } ) |
  ( const entries = { <entriesList> } ) |  /* immutable entries */
  ( { const } <nonTableKwName> = <expression> ; )

<keyElementList> := { <keyElement> }*

<keyElement> := <expression> : <name> ;

<actionList> := { <actionRef> ; }*

<actionRef> := <nonTypeName> { ( <argumentList> ) }

<entriesList> := <entry> { <entry> }*

<entry> := <keysetExpression> : <actionRef> ;

<actionDeclaration> := action <name> ( <parameterList> ) <blockStatement>
```

VARIABLES

```
<variableDeclaration> := { const } <typeRef> <name> { = <expression> };
```

EXPRESSIONS

```
<functionDeclaration> := <functionPrototype> <blockStatement>

<argumentList> := { <argument> { , <argument> }* }

<argument> := <expression> | _

<expressionList> := { <expression> { , <expression> }* }

<lvalue> := <nonTypeName> {
    ( . <name> ) |  /* member selector */
    ( [ <indexExpression> ] )  /* array subscript */
  }*

<expression> := <expressionPrimary> { <exprOperator> <expression> }*

<expressionPrimary> := <integer> | <boolean> | <string> |
  ( ~~{ . }~~ <nonTypeName> ) |
  ( { <expressionList> } ) |
  ( ( <expression> ) ) |
  ( ( ! | ~ | - ) <expression> ) |  /* unary expression */
  ( TYPE_IDENTIFIER | error ) |  /* member selector, function call, constructor */
  ( ( <typeRef> ) <expression> )  /* cast */

<exprOperator> := <binaryOperator> |
  ( . <name> ) |  /* member selector */
  ( [ <indexExpression> ] ) |  /* array subscript */
  ( ( <argumentList> ) ) |  /* function call */
  ~~( < <realTypeArgumentList> > )~~ |
  ( = <expression> )  /* named argument */

<indexExpression> := <expression> { : <expression> }

<integer> := INTEGER

<boolean> := true | false

<string> := STRING

<binaryOperator> := * | / | + | - | <= | >= | < | > | != | == | || | && | | | & | << | >>
```

Syntax Tree

PROGRAM

<p4program> := <declarationList>$_{decl\_list}$

<declarationList> := { <declaration>$_{[0..n]}$ }*

<declaration> := (<variableDeclaration> | <externDeclaration> | <actionDeclaration> |
  <functionDeclaration> | <parserDeclaration> | <parserTypeDeclaration> | <controlDeclaration> |
  <controlTypeDeclaration> | <typeDeclaration> | <errorDeclaration> | <matchKindDeclaration> |
  <instantiation>)$_{decl}$

<name> := STRING$_{strname}$

<parameterList> := { <parameter>$_{[0..n]}$ }*

<parameter> := DIRECTION$_{direction}$ <typeRef>$_{type}$ <name>$_{name}$ { <expression> }$_{init\_expr}$

<packageTypeDeclaration> := <name>$_{name}$ ~~{ <typeParameterList> }~~$_{~~type\_params~~}$ <parameterList>$_{params}$

<instantiation> := <typeRef>$_{type}$ <argumentList>$_{args}$ <name>$_{name}$

PARSER

<parserDeclaration> := <typeDeclaration>$_{proto}$ { <parameterList> }$_{ctor\_params}$
  <parserLocalElements>$_{local\_elements}$ <parserStates>$_{states}$

<parserTypeDeclaration> := <name>$_{name}$ ~~{ <typeParameterList> }$_{type\_params}$~~ <parameterList>$_{params}$

<parserLocalElements> := { <parserLocalElement>$_{[0..n]}$ }*

<parserLocalElement> := (<variableDeclaration> | <instantiation>)$_{element}$

<parserStates> := { <parserState>$_{[0..n]}$ }+

<parserState> := <name>$_{name}$ <parserStatements>$_{stmt\_list}$ <transitionStatement>$_{transition\_stmt}$

<parserStatements> := { <parserStatement>$_{[0..n]>}$ }*

<parserStatement> := (<assignmentStatement> | <functionCall> | <directApplication> |
  <parserBlockStatement> | <variableDeclaration>)$_{stmt}$

<parserBlockStatement> := <parserStatements>$_{stmt\_list}$

<transitionStatement> := <stateExpression>$_{stmt}$

<stateExpression> := (<name> | <selectExpression>)$_{expr}$

<selectExpression> := <expressionList>$_{expr\_list}$ <selectCaseList>$_{case\_list}$

<selectCaseList> := { <selectCase>$_{[0..n]}$ }*

<selectCase> := <keysetExpression>$_{keyset\_expr}$ <name>$_{name}$

<keysetExpression> := (<tupleKeysetExpression> | <simpleKeysetExpression>)$_{expr}$

<tupleKeysetExpression> := <simpleExpressionList>$_{expr\_list}$

<simpleKeysetExpression> := (<expression> | <default> | <dontcare>)$_{expr}$

<simpleExpressionList> := { <simpleKeysetExpression>$_{[0..n]}$ }+

CONTROL

<controlDeclaration> := <typeDeclaration>$_{proto}$ { <parameterList> }$_{ctor\_params}$
  <controlLocalDeclarations>$_{local\_decls}$ <blockStatement>$_{apply\_stmt}$

<controlTypeDeclaration> := <name>$_{name}$ ~~{ <typeParameterList> }$_{type\_params}$~~ <parameterList>$_{params}$

<controlLocalDeclarations> := { <controlLocalDeclaration>$_{[0..n]}$ }*

<controlLocalDeclaration> := (<variableDeclaration> | <actionDeclaration> | <tableDeclaration> |
  <instantiation>)$_{decl}$

EXTERN

<externDeclaration> := (<externTypeDeclaration> | <functionPrototype>)$_{decl}$

<externTypeDeclaration> := <name>$_{name}$ ~~{ <typeParameterList>$_{type\_params}$ }~~ <methodPrototypes>$_{method\_protos}$

<methodPrototypes> := { <functionPrototype>$_{[0..n]}$ }*

<functionPrototype> := { <typeRef> }$_{return\_type}$ <name>$_{name}$ ~~{ <typeParameterList> }$_{type\_params}$~~
  <parameterList>$_{params}$

TYPES

<typeRef> := (<baseTypeBoolean> | <baseTypeInteger> | <baseTypeBit> | <baseTypeVarbit> |
  <baseTypeString> | <baseTypeVoid> | <baseTypeError> | <name> | ~~<specializedType> |~~
  <headerStackType> | <tupleType>)$_{type}$

<tupleType> := <typeArgumentList>$_{type\_args}$

<headerStackType> := <typeRef>$_{type}$ <expression>$_{stack\_expr}$

~~<specializedType> := <typeRef>$_{type}$ <typeArgumentList>$_{type\_args}$~~

<baseTypeBoolean> := <name>$_{name}$
<baseTypeInteger> := <name>$_{name}$ { <integerTypeSize> }$_{size}$
<baseTypeBit> := <name>$_{name}$ { <integerTypeSize> }$_{size}$
<baseTypeVarbit> := <name>$_{name}$ <integerTypeSize>$_{size}$
<baseTypeString> := <name>$_{name}$
<baseTypeVoid> := <name>$_{name}$
<baseTypeError> := <name>$_{name}$

<integerTypeSize> := INTEGER$_{size}$

~~<typeParameterList> := { <name>$_{[0..n]}$ }+~~

<realTypeArg> := (<typeRef> | <dontcare>)$_{arg}$

<typeArg> := (<typeRef> | <name> | <dontcare>)$_{arg}$

<realTypeArgumentList> := { <realTypeArg>$_{[0..n]}$ }+

<typeArgumentList> := { <typeArg>$_{[0..n]}$ }*

<typeDeclaration> := (<derivedTypeDeclaration> | <typedefDeclaration> | <parserTypeDeclaration> |
  <controlTypeDeclaration> | <packageTypeDeclaration>)$_{decl}$

<derivedTypeDeclaration> := (<headerTypeDeclaration> | <headerUnionDeclaration> |
  <structTypeDeclaration> | <enumDeclaration>)$_{decl}$

<headerTypeDeclaration> := <name>$_{name}$ <structFieldList>$_{fields}$

<headerUnionDeclaration> := <name>$_{name}$ <structFieldList>$_{fields}$

<structTypeDeclaration> := <name>$_{name}$ <structFieldList>$_{fields}$

<structFieldList> := { <structField>$_{[0..n]}$ }*

<structField> := <typeRef>$_{type}$ <name>$_{name}$

<enumDeclaration> := INTEGER$_{type_size}$ <name>$_{name}$ <specifiedIdentifierList>$_{fields}$

<errorDeclaration> := <identifierList>$_{fields}$

<matchKindDeclaration> := <identifierList>$_{fields}$

<identifierList> := { <name>$_{[0..n]}$ }+

<specifiedIdentifierList> := { <specifiedIdentifier>$_{[0..n]}$ }+

<specifiedIdentifier> := <name>$_{name}$ { <expression> }$_{init\_expr}$

<typedefDeclaration> := (<typeRef> | <derivedTypeDeclaration>)$_{type\_ref}$ <name>$_{name}$

STATEMENTS

<assignmentStatement> := (<expression> | <lvalueExpression>)$_{lhs\_expr}$ <expression>$_{rhs\_expr}$

<functionCall> := (<expression> | <lvalueExpression>)$_{lhs\_expr}$ <argumentList>$_{args}$

<returnStatement> := { <expression> }$_{expr}$

<exitStatement> := **exit**

<conditionalStatement> := <expression>$_{cond\_expr}$ <statement>$_{stmt}$ { <statement>$_{else\_stmt}$ }

<directApplication> := (<name> | <typeRef>)$_{name}$ <argumentList>$_{args}$

<statement> := (<assignmentStatement> | <functionCall> | <directApplication> |
  <conditionalStatement> | <emptyStatement> | <blockStatement> | <exitStatement> |
  <returnStatement> | <switchStatement>)$_{stmt}$

<blockStatement> := <statementOrDeclList>$_{stmt\_list}$

<statementOrDeclList> := { <statementOrDeclaration>$_{[0..n]}$ }*

<switchStatement> := <expression>$_{expr}$ <switchCases>$_{switch\_cases}$

<switchCases> := { <switchCase>$_{[0..n]}$ }*

<switchCase> := <switchLabel>$_{label}$ { <blockStatement>$_{stmt}$ }

<switchLabel> := (<name> | <default>)$_{label}$

<statementOrDeclaration> := (<variableDeclaration> | <statement> | <instantiation>)$_{stmt}$

TABLES

<tableDeclaration> := <name>$_{name}$ <tablePropertyList>$_{prop\_list}$

<tablePropertyList> := { tableProperty$_{[0..n]}$ }+

<tableProperty> := (<keyProperty> | <actionsProperty> | <entriesProperty> | <simpleProperty>)$_{prop}$

<keyProperty> := <keyElementList>$_{keyelem\_list}$

<keyElementList> := { <keyElement>$_{[0..n]}$ }*

<keyElement> := <expression>$_{expr}$ <name>$_{match}$

<actionsProperty> := <actionList>$_{action\_list}$

<actionList> := { <actionRef>$_{[0..n]}$ }*

<actionRef> := <name>$_{name}$ { <argumentList>$_{args}$ }

<entriesProperty> := <entriesList>$_{entries\_list}$

<entriesList> := { <entry>$_{[0..n]}$ }+

<entry> := <keysetExpression>$_{keyset}$ <actionRef>$_{action}$

<simpleProperty> := <name>$_{name}$ <expression>$_{init\_expr}$

<actionDeclaration> := <name>$_{name}$ <parameterList>$_{params}$ <blockStatement>$_{stmt}$

VARIABLES

<variableDeclaration> := <typeRef>$_{type}$ <name>$_{name}$ { <expression> }$_{init\_expr}$

EXPRESSIONS

<functionDeclaration> := <functionPrototype>$_{proto}$ <blockStatement>$_{stmt}$

<argumentList> := { <argument>$_{[0..n]}$ }*

<argument> := (<expression> |)$_{arg}$

<expressionList> := { <expression>$_{[0..n]}$ }*

<lvalueExpression> := (<name> | <memberSelector> | <arraySubscript>)$_{expr}$

<expression> := (<expression> | <booleanLiteral> | <integerLiteral> | <stringLiteral> | <name> |
<expressionList> | <castExpression> | <unaryExpression> | <binaryExpression> | <memberSelector> |
<arraySubscript> | <functionCall>)$_{expr}$ ~~{ <realTypeArgumentList> }$_{type\_args}$~~

<castExpression> := <typeRef>$_{type}$ <expression>$_{expr}$

<unaryExpression> := OPERATOR$_{op}$ <expression>$_{operand}$

<binaryExpression> := <expression>$_{left\_operand}$ OPERATOR$_{op}$ <expression>$_{right\_operand}$

<memberSelector> := (<expression> | <lvalueExpression>)$_{lhs\_expr}$ <name>$_{name}$

<arraySubscript> := (<expression> | <lvalueExpression>)$_{lhs\_expr}$ <indexExpression>$_{index\_expr}$

<indexExpression> := <expression>$_{start\_index}$ { <expression> }$_{end\_index}$

<booleanLiteral> := INTEGER$_{value}$

<integerLiteral> := INTEGER$_{value}$ INTEGER$_{width}$

<stringLiteral> := STRING$_{value}$

<default> := **default**

<dontcare> := _